

Exploring Practical Limitations of TCP Over Transatlantic Networks

Antony Antony^a, Johan Blom^b, Cees de Laat^b, Jason Lee^{b,c}

^a*NIKHEF, 409 Kruislaan, 1098 SJ Amsterdam, The Netherlands*

^b*Universteit van Amsterdam, 403 Kruislaan, 1098 SJ Amsterdam, The Netherlands*

^c*Lawrence Berkeley National Laboratory, Berkeley, CA, USA*

Abstract

Tomorrow's large physics and astronomy projects will require to transport tremendous amounts of data over long distances in near real time. Traditional TCP implementations have severe problems in reaching the necessary performance. In the recent past, researchers have shown that TCP implementations can be scaled to achieve multi-gigabit per second speeds over high-bandwidth high-delay networks. The ability of TCP to scale to high speeds opens possibilities for very large data transfers over vast distances. We analyze here whether TCP can fulfill this task and what problems we are faced with. We also examine TCP in the context of dedicated links (Lambdas).

Key words: TCP, TCP response behavior, HSTCP, Grid networking

1 Introduction

Many large physics and astronomy projects that are coming online in the next couple of years will require tremendous amounts of bandwidth over long distances[24]. Common TCP implementations show many problems when trying to achieve good throughput in a high bandwidth long delay environment. Fortunately, network researchers have recently shown that TCP implementations can be scaled to achieve multi-gigabit per second speeds over high-

bandwidth high-delay networks[8][3]. The ability of TCP to start scaling to these high speeds opens possibilities for large data transfers over long distances. Two factors make this a challenging problem. First, the distance being traversed is greater than 6000 kilometers (which translates into a packet delay greater than 100 ms). Second, the desired throughput exceeds 1 Gigabit per second (Gbit/s). The combination of these two factors is commonly called the Bandwidth Delay Product (BDP), and traversing networks with a large BDP is known to be a difficult problem in networking[3]. In general it is groups of network engineers or "wizards" [13] working together to create ideal conditions for TCP who have achieved high bandwidth on these links[20]. The problem with this is

Email addresses: antony@nikhef.nl (Antony Antony), jblom@science.uva.nl (Johan Blom), delaata@science.uva.nl (Cees de Laat), JRLee@lbl.gov (Jason Lee).

that most day to day situations do not present such ideal conditions, and the average user is not usually a network wizard but perhaps a physicist or an astronomer. In this paper we show how and why TCP performs so well under ideal situations and what happens under less than ideal conditions.

In this paper, we present the test results of various modifications to the TCP algorithms and various host parameters to achieve good throughput under non-ideal situations. We examine TCP in depth and try to explain why TCP is more sensitive to particular conditions that arise in large BDP networks and why one is more likely, now than in the past, to experience these conditions in a day-to-day usage. Then we analyze how far TCP can scale, and explore what problems this may cause in the future. The rest of this paper is organized as follows. In Section 2, we show related work on different TCP implementations and modified algorithms. In Section 3, we show the experimental setups used in the experiments described in section 6. In Section 4 we explore adaptations to TCP that can be done, while section 5 delves into the reasons why current implementations experience problems. Section 7 discusses the results. Finally, we present concluding remarks in Section 8.

2 Related Work

Recently, the network research community has proposed changes to TCP to help it cope with the latest advances in networking. In this section we first describe a new trend in networking called lambda networking. Then we give a brief overview of standard TCP and its main variants.

2.1 Lambda Networking

A new paradigm in high-bandwidth high-delay networking is the trend towards provisioning end-to-end light paths for sender, receiver pairs, called *Lambdas*. Since Lambdas are end-to-end paths, there should be little or no cross traffic on the link; yet TCP often experiences “congestion signals”, which cause TCP to under-utilize the available bandwidth. It is important for Lambda users to have a high average utilization of the path. Our tests using current TCP algorithms show that TCP needs to be able to overcome this congestion on a Lambda and fully utilize the network. If TCP is not capable of this, users will often turn to other more aggressive transport protocols such as Tsunami [21] or Sabul[22].

2.2 Standard TCP

Standard TCP is the RFC793 [14] and RFC 2581 [15] compliant version of the TCP stack. During the initial start up phase (*slow start*), TCP sends out two packets, and for each packet acknowledgment received, sends out two packets. This exponentially increases the amount of data being sent until reaching either the capacity of the sending host or the network capacity. TCP assumes that it has reached this capacity when it does not receive an acknowledgment for a packet. This loss causes TCP to back off, which is done by decreasing the number of packets that can be in flight and going into *congestion avoidance*. The standard formula for computing this decrease is to multiply by 0.5 the number of packets that are currently unacknowledged. This is called the Multiplicative Decrease (MD) and the value of 0.5 cuts the number of

packets in flight in half. TCP is now in what is known as the congestion avoidance phase, and will only send more packets every RTT (Additive Increase, AI). This effectively means that when all outstanding packets have been acknowledged, it will then add one more packet to the number of packets that can now be outstanding. This is why TCP’s congestion control is commonly referred to as being an AIMD algorithm. TCP’s response function $w = 1.2/\sqrt{p}$, where p is the steady state drop rate [25], is derived from the AIMD parameters.

The problem with standard TCP over very high BDP networks is that the number of packets in flight can be very large, and the time that it takes to recover from a congestion event is directly proportional to the BDP. This means that TCP is not scale invariant with respect to bandwidth. For example, if we have a 200 ms path, with a capacity of 1 Gbit/s, and sh a congestion event, it will take at least 28 minutes to recover from a single congestion event (based on a standard packet size of 1500 bytes).

2.3 TCP variants

HSTCP: High Speed TCP is an enhancement to TCP’s congestion avoidance algorithm that has been recently proposed by Sally Floyd [25]. HSTCP adjusts the AI and MD parameters. HSTCP uses a table of values based on the number of currently outstanding packets to figure out what the AI and MD parameters should be. The values for AI range from 1 (standard TCP) to a high of 73 packets, and the range of MD is from 0.5 (standard TCP) to a low of 0.09. Consequently, when a congestion event occurs over large BDP networks, TCP does not drop back as

much and adds more than one packet per RTT, thus recovering faster.

Scalable TCP: Instead of looking for the best values for AI and MD for a given network, scalable TCP [8] takes the approach that MD should always be .125 (reduce window by 1/8 on congestion events). Also, instead of doing an additive increase, it does a *multiplicative increase* beyond a certain threshold. The multiplicative increase is of 5% of the current congestion window. This has the effect of forcing TCP to always recover in a fixed number of RTTs. In the example above where we used a 200ms link, Scalable TCP’s recovery time is approximately 2.7 seconds.

Fast TCP: Fast TCP [23] modifies what TCP considers to be a congestion event by using queue delay combined with packet loss to decide if it should go into congestion avoidance. It also includes modifications to the TCP sender to pace out packets so as to not overwhelm the network. Finally it makes some changes to congestion avoidance so that it recovers more quickly after a congestion event.

3 Experimental Setup

In order to illustrate the behavior of TCP on large BDP networks we show results gathered in two networks with different characteristics. These networks were chosen because they met all the requirements for large BDP networks: they are high-bandwidth, high-latency networks with end-to-end bandwidth provisioned in excess of 1 Gbit/s (gigabit Ethernet in STS-24) and a latency greater than 100 ms. We had control over the network path and the end hosts, including the host hardware,

kernels, and all routers along the path. This allowed us to monitor in depth the interactions of TCP with each element of these testbeds, and also gave us the ability to induce rate limiting into the path and observe its effects.

All the networks used in our tests were *over-provisioned*, meaning that the network could never be congested simply by the output of the computers used for testing. As a result, for there to be congestion on the links, it would have to be artificially introduced. Over-provisioned networks are common; e.g. many Internet backbones are over-provisioned, running at rates of 1 Gbit/s to 10 Gbit/s.

Our tests were conducted in two testbeds depicted in Figure 1(a) and Figure 1(b). The first network is the DataTAG testbed [1], a transatlantic testbed that was created to demonstrate the issues in connecting Grids separated by great distances. The DataTAG testbed spans over 6000 km, with an average round trip time (RTT) of 120 ms. The network backbone is constructed from an STS-16, which provides 2.5 Gbit/s of bandwidth between the two sites. The hosts located in Europe are at CERN [2] in Geneva, Switzerland, and the hosts located in the U.S. are at StarTAP [17] in Chicago. This path is routed via SURFnet Amsterdam, and is a Lambda (optical, no routers in the middle) of 1 Gbit/s provisioned inside an OC192 (10 Gbit/s) link.

The second network is composed of hosts located at NIKHEF and SARA in Amsterdam, The Netherlands. The link crosses the Atlantic, is “soft looped” at StarLight in a Time Division Multiplexing (TDM) switch, and returns to The Netherlands, for a total latency of 218 ms. This latency is roughly double that

of the DataTAG testbed. This link is also provisioned over an OC-192, providing 10 Gbit/s of bandwidth.

Both paths are provisioned inside an OC192 SONET wide area link interconnected via a TDM switch, Cisco ONS 15454. In our setup it is possible to provision an end-to-end light path of various speeds such as 1 Gbit/s or 622 Mbit/s. When the speed is 622 Mbit/s the path has a bottleneck at the TDM switch. The buffer memory available at the bottleneck is 512 kBytes.

During our tests, these two testbeds were isolated from all other traffic, to rule out external influences on the links or routers. Note that the network diagrams in Fig 1(a) and Fig 1(b) only show the physical path that the packets took and do not depict connections between these testbeds and external networks. They should not be taken as complete diagrams of the DataTAG and SURFnet networks.

The PCs used in the DataTAG testbed were running the Linux kernel version 2.4 with Net100 enhancements [6] to allow instrumentation of the TCP stack internals. The hardware configuration of the PCs was:

- Dual Intel(R) XEON(TM) CPU 2.40 GHz with:
 - 1 Gigabyte of memory
 - PCI-X, 64 bits 66 MHz bus
 - SysKonnect SK-NET Gigabit Ethernet Adapter SK-9843 SX
 - Intel(R) PRO/1000

These were set with the command ‘sysctl -p’.

```
net.core.rmem max = 33554432
net.core.wmem max = 33554432
net.core.rmem default = 65536
net.core.wmem default = 65536
net.ipv4.tcp_rmem = 4096 87380 33554432
net.ipv4.tcp_wmem = 4096 65536 33554432
net.ipv4.tcp_mem = 33554432 33554432 33554432
net.core.mod cong = 2800
```

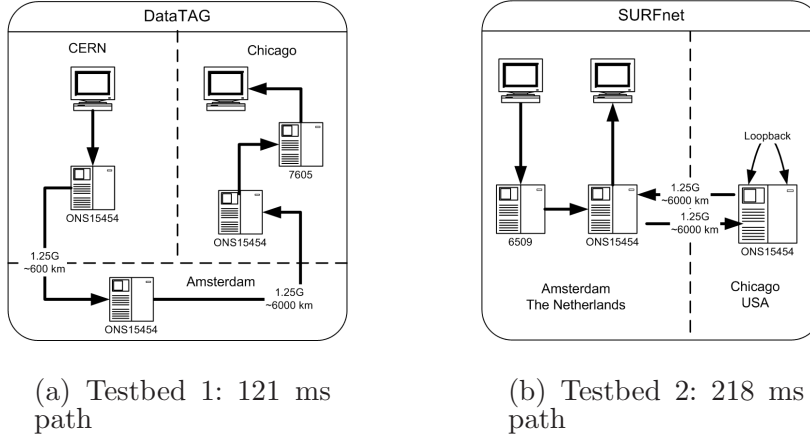


Fig. 1. Testbed setup.

```
net.core.low cong = 1000
net.core.no cong = 200
net.core.cong thresh = 2900
net.ipv4.route.flush = 1
```

In all of our tests the machines used to drive the network were capable of generating several Gbit/s, ensuring that the end-hosts themselves would never be the bottleneck. This is very important, as the profiling of TCP in cases where the end-host (including the NIC) is the bottleneck, is vastly different from cases where the network is the bottleneck. In cases where the end-host is the bottleneck, it appears to the TCP congestion control algorithm that there is a consistently congested network, even if there is little or no actual congestion on the link; TCP reacts by reducing its output and offering low throughput to the user, even if there is excess network capacity.

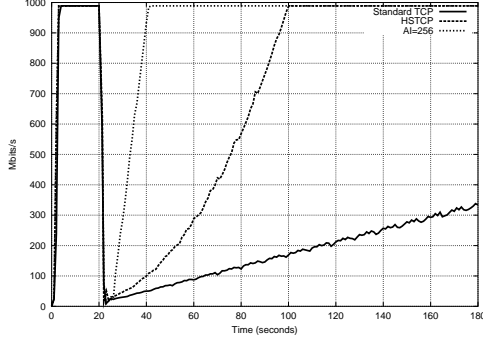
4 TCP and Beyond

In this section we explore the parameters that can be modified to optimize the throughput of TCP in different networks using Lambdas or not. On a large BDP path, changing the parameters of the AI and

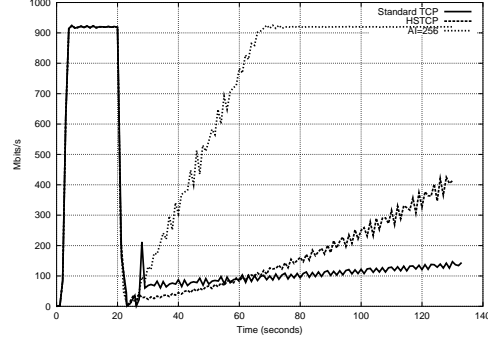
AI	MD		
	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$
1	28	19	14
2	14	9.25	7
4	7	6	3.5
8	2.5	4	1.25
16	1.25	0.5	0.625

Table 1
Recovery time for various AIMD Values in minutes

MD algorithms has a dramatic effect on the time to recover from a single congestion event. In Table 1, we calculate the theoretical time to recover from a single congestion event on a 100 ms RTT link with a capacity of 1 Gbit/s. These numbers can be calculated for different paths by simply multiplying by the ratio of the BDPs; e.g., on a path with twice the latency you should just double the times, and on a path with a capacity of only 500 Mbit/s you should divide the values in half. Of course, recovering faster from a congestion event improves the overall throughput (for one application). Therefore, the ability to tune the parameters of the AI and MD algorithms may help many network applications. However, most TCP implementations do not provide control over these parameters.



(a) Testbed 1



(b) Testbed 2

Fig. 2. TCP response behavior to a severe congestion over 1 Gbit/s path: Throughput vs Time.

In Figure 2 we plotted the response to server congestion of three TCP variants: HSTCP, standard TCP, and TCP with an AI of 256 packets. The best response came from TCP with an AI value of 256. Unfortunately, this strategy has the side effect of overshooting the capacity of the network by a large number of packets, causing massive retransmits and restricting the overall bandwidth. It also is very unfriendly to other users of the link.

However, we believe that for future end-to-end light path based networks (Lambdas), it is essential to allow some control over the values of AI and MD. Ideally these values should be controllable on a per flow basis, perhaps based on source/destination IP addresses and ports or some other metrics (VLAN tagging, TOS, etc.). The Net100[6] project is an example of how to tune the AIMD parameters for individual network flows. The reason that these values need to be controlled on a per flow basis is that many of the modifications we are talking about are not “fair” to other users of the network, and therefore need to be restricted to only certain flows.

A major requirement for TCP is fairness. Fairness is most relevant in cases where several competing TCP flows share the same network path and the network has to merge this traffic together. If one of the flows does not back off, it can starve the other flows. We acknowledge that fairness is essential to prevent the collapse of shared networks. Yet, on Lambdas (end-to-end, single use, isolated networks, where there is no other traffic) there is a need to allow users to quickly obtain and maintain a high utilization of the network in spite of spurious congestion events. It should be noted that in none of the tests are we testing TCP against itself or other variants of TCP. We are simply characterizing the behavior of TCP and its variants in the presence of congestion events.

5 Barriers to TCP

There are many reasons for users to have trouble achieving gigabit per second speeds across their networks. Most of the reasons can be tracked back to “congestion signals” in TCP. What causes congestion signals in TCP can be traced back to either

cross traffic (as is encountered in the Internet) or hardware problems. On isolated network paths we often see congestion from hardware. In this section, we review TCP congestion and then some of the sources of the hardware problems.

5.1 TCP congestion

TCP congestion control as it is described in RFC 2581 [15], is not tied to a particular hardware architecture or implementation, and it does not take into account restrictions that may be imposed by the network on the flow. Such restrictions occur because the network is composed of scarce resources such as buffers in the hosts, routers and network interface cards (NICs). Once these resources are oversubscribed, problems arise such as the dropping of packets from queues in routers (not to mention QoS effects like Random Early Discard and Weighted Random Early Discard). Details of the protocol implementation and end-host hardware can also have a significant effect on TCP's performance. The performance is influenced by factors such as the inter-packet gap at physical layers, which can in turn affect how TCP interacts with itself and other protocols as several flows are multiplexed together into a single channel at a backbone router.

We have found that in practice the main factors influencing the performance of a TCP flow are the lower layer characteristics. A flow has to share the available resources on both end systems as well as on all network elements along the path. TCP implementations and network topologies are the true limits of TCP, not the protocol. The basic problem is that if there is congestion on the network or if TCP interprets an event as mean-

ing congestion, it becomes impossible to obtain even close to gigabit speeds on paths where the RTT exceeds 80 ms. This is mostly due to the congestion avoidance algorithm. Even in cases where RTT is less than 80 ms, there are still many factors which can adversely influence performance. For example, the standard timers on most operating systems do not have fine enough granularity to time out the packets correctly. Thus, the input and output queues on the NICs that handle multiple flows by servicing interrupts at high rates are usually approximations of what the protocol requires. Details needing careful attention are: host issues (CPU, bus, memory), transmit queues on NICs, interrupt moderation on sender and receiver (reducing the load arriving packets).

5.2 Hardware problems

The paradigm whereby the network is seen as a black box, data goes in one side and out the other side, and all the complexity in between (routers, switches, etc.) is transparent is simply wrong. This black box paradigm has severe limitations in high bandwidth high delay networks, particularly with respect to the granularity of the timers. The TCP protocol is described in the context of timers, and has a concept of timing out packets based on timers. Thus, the vision of the network and hosts as black boxes fails, since they have interactions that change the behavior of packets while they are in transit. Exposing the host and network internals is also difficult. Take for example a high throughput infrastructure using two PCs separated by a large RTT. Even assuming an ideal network with no loss, the problem is still hitting the end-hosts. Different NICs perform differently, and how

they burst (and buffer) out packets has performance effects. If one end-host can over-run the other host with packets, performance suffers. There is also the problem that if the host is doing other activities such as computation or disk access, we can expect a drop in TCP performance. Even if we have extra CPU power, it only takes a fraction of a second for the CPU to be working on something else and miss servicing the interrupt in time to keep TCP flowing correctly.

6 Experiments

To validate how TCP reacts (i.e. its response function) to congestion in the network (i.e. a bottleneck link), we took the two testbeds and artificially lowered the link capacity. For both networks, we reduced the maximum throughput of the link to 622 Mbit/s, while leaving the NICs on the the host machines capable of sending at 1 Gbit/s. This allowed TCP to overrun the capacity of the network, thereby showing how TCP adjusts to use the available bandwidth of the link. For all the tests we reconfigured the host for optimal performance on this link. In Figure 3(b) we show results for the physical topology of a series of tests with standard TCP and varying streams and window sizes.

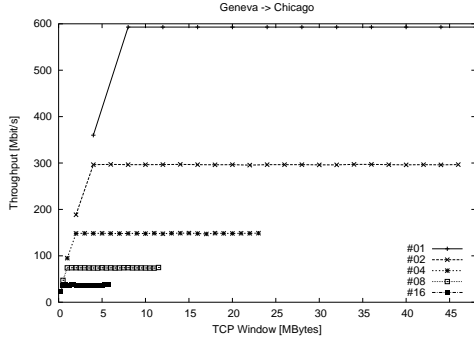
All the streams cluster around a 30 Mbit/s throughput. This poor performance occurs because, due to perceived congestion, the streams are unable to complete slow start. The streams fall out of slow start early in the transfer, giving each stream a low throughput.

In examining each of the streams throughput over time, we see that their congestion windows never use

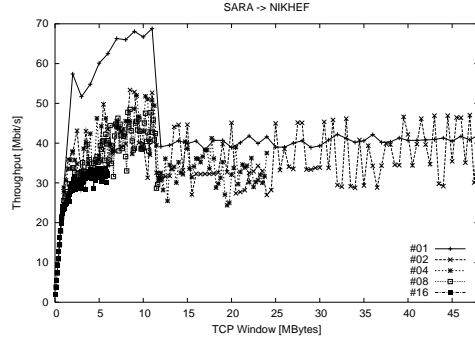
the buffer space allocated to them. The reduction of the congestion window suggests that there is some form of congestion on the link (the under-provisioned link of 622 Mbit/s) and that a single stream will never attain and maintain good link utilization. This is a common scenario on high-speed WANs.

In the multi-stream runs, one can see that the link is capable of supporting 16 concurrent flows, each with a 30 Mbit/s throughput, giving a total link utilization of 77 %, which is much better than the 1 % achieved with a single stream. Given this baseline for TCP, we then tested how different TCP variants react to congestion.

We ran several modified TCP stacks (HSTCP, TCP with a large AI value) on this network. The results were surprising (see Fig 4(b)). Notice that there are now severe oscillations in the bandwidth. This is because the TCP variants keep ramping up their bandwidth until they reach the NIC speed (1 Gbit/s) which then overruns the link speed (622 Mbit/s) and causes packet loss. Packet loss in TCP is a congestion event which causes TCP to back-off, enter congestion avoidance and then start the whole cycle again. The reason oscillations are so severe in testbed 2 as compared to testbed 1 is that the input buffer on the ONS (see Fig 1(b)) is too small for the packet bursts that come out of the NIC on the sending host. In this example one can see that each of the other modified TCP stacks also experiences the same problem. Each of them reaches link capacity and then experiences packet loss and drops into congestion avoidance phase (Fig. 4). After a congestion event is received, it takes almost 20 s for TCP to stabilize its algorithm enough to try raising the congestion window. During the

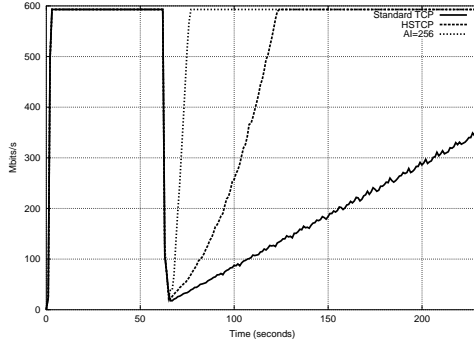


(a) Testbed 1

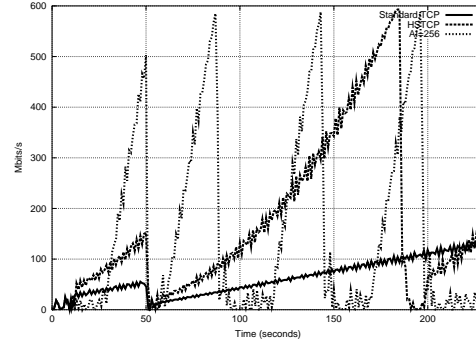


(b) Testbed 2

Fig. 3. Throughput vs time over 622 Mbit/s bottleneck link.



(a) Testbed 1



(b) Testbed 2

Fig. 4. TCP response behavior for a 622 Mbit/s path: Throughput vs Time.

pause between the congestion event and the start of recovery, TCP has to wait for all outstanding packets to be acknowledged or for them to time out. These retransmissions and time outs cause very long pauses in the data transfer.

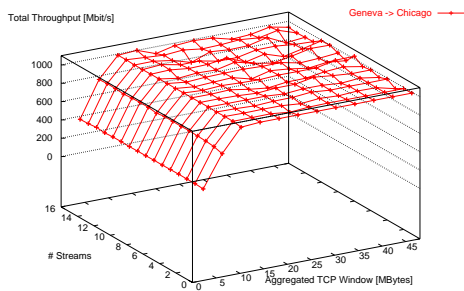
The examples above point out some of the flaws in TCP's congestion avoidance and recovery mechanisms. They also show that while HSTCP and TCP with an AI of 256 does not make it worse, yet they also do not address this issue. There is still work to be done on scaling TCP to large BDP networks where congestion is present. The answer up to now has always been to over-provision the

network, but in the future this may not always be an option with 10 Gbit/s NICs coming on the market.

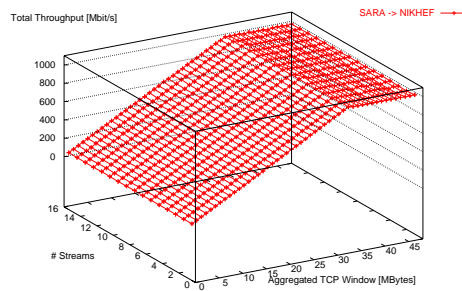
7 Using the Testbeds

In this section we analyze what is the best set of parameters to use for each testbed (i.e. the number of streams and sending congestion window size) and show that the best combination is not always the most intuitive.

In cases where a single TCP stream is unable to obtain the full bandwidth of a link, using multiple streams can



(a) Testbed 1



(b) Testbed 2

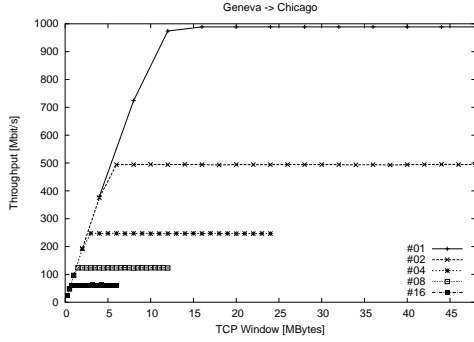
Fig. 5. Throughput vs. Number of streams vs Window size over 1 Gbit/s path

sometimes improve the situation. There are two reasons for this. First, the aggregate congestion window is larger with several streams than with a single stream. Second, in the case of a spurious congestion event, only one of the TCP streams is affected, thus mitigating the consequences of a congestion event to $1/N$, where N is the number of parallel streams. The problem with this approach is that it can often complicate the data transfer itself. The sender has to fragment the data across the streams, and the receiver has to reassemble them. All of this must be done with user-level operating system calls, as there is no support for this in current TCP stacks. Several packages such as Psockets [12] can help with this, but these packages cannot eliminate the overhead of fragmenting and reassembling the data, which can become significant at gigabit speeds.

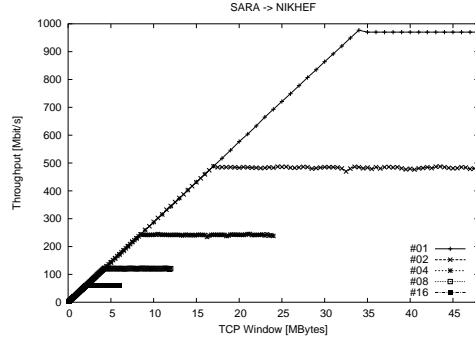
Not allocating a large enough buffer for the sender’s congestion window is a common problem in tuning scenarios. Figure 5(a) and Figure 5(b) show that, in both testbeds, setting a small TCP buffer for the window size will prevent TCP from attaining full utilization of the link. A buffer that is too large can also lower TCP’s performance, as well as wasting system

resources. In Figure 5(a) one can see that as we continue to increase window buffer sizes, even after achieving full bandwidth, TCP starts fluctuating and the bandwidth becomes less steady. In Figure 5(b), this fluctuation is less pronounced. The reason for this is that the sender’s packet bursts have been “spread out” by a larger number of intervening routers, which are designed to evenly pace packets out.

In Figure 5(a) we can see that, in all tests, simply having a window size greater than 14 MBytes was sufficient to saturate the link in Testbed 1; on Testbed 2 a TCP window greater than 29 MBytes had the same effect as shown in Figure 5(b). Figure 6 shows the breakdown, by congestion window size, of the average speed of the individual streams on the two testbeds. In the single stream case, one can see that with a window size of 14 MBytes on Testbed 1 (with the 100 ms latency path) line speed can be obtained (989 Mbit/s). On Testbed 2 we do not achieve full link utilization until the congestion window reaches 32 MBytes. Given that Testbed 2 has twice the latency (and thus BDP) of Testbed 1, this congestion window size (29 MBytes plus TCP and Linux kernel overhead



(a) Testbed 1



(b) Testbed 2

Fig. 6. Average throughput per/stream vs. TCP socket size over 1 Gbit/s path

and rounding) is just what we would expect.

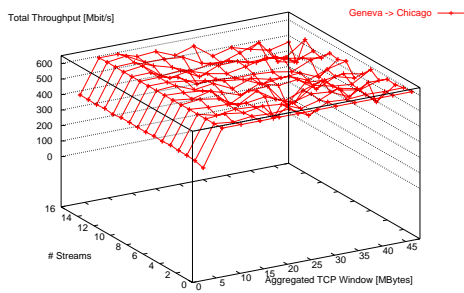
In Figure 7, we modified the experiment performed for Figure 5 by throttling the link from 1.25 Gbit/s to 622 Mbit/s. Note that in Testbed 1, as soon as we have a congestion window of 8 MBytes, we can fully utilize the link, but unlike before, adding more streams and/or growing the congestion window does not help. In fact, if we try to utilize 16 concurrent streams, we suffer a 20 % drop in our link utilization. The reason for this is that the sending host can send faster than the bottleneck, thus each of our streams is ramping up until a congestion event occurs. When this happens, the stream drops into congestion avoidance, but its burst has caused a congestion event on the other streams, causing them to drop into congestion avoidance, which results in an overall lower throughput for all the streams.

We have further detailed the results for a varying number of streams in Figure 3(a) and Figure 3(b). Figure 3(a) shows a fairly consistent pattern. With one stream we can attain a bandwidth of approximately 592 Mbit/s, i.e. a link utilization in excess of 95 %! With two streams,

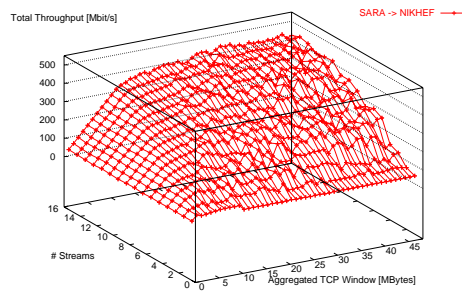
each of the streams is able to reach about 300 Mbit/s, again an amazing utilization of the link. In Figure 3(b), however, one notices that the pattern seen in Testbed 1 is missing. Instead we have a much more chaotic representation of the data. A single stream never reaches more than 60 Mbit/s, i.e. only 10 % utilization, and on average all streams are only capable of reaching a bandwidth of about 40 Mbit/s no matter what their congestion window size is. In this case it is only possible to get full link utilization by using all 16 streams.

8 Conclusion

In this paper we have shown that TCP is not intrinsically limited by its congestion control algorithms, but rather by its implementations. We showed that there are ways to scale TCP for large BDP networks, and described what problems can arise while doing this. We have examined the properties of end-hosts and the characteristics of the networks and showed how these can influence TCP's behavior. We have also explained that TCP can be adapted to scale to large BDP networks with



(a) Testbed 1



(b) Testbed 2

Fig. 7. Throughput vs time over 622 Mbit/s bottleneck link.

a minor amount of tuning. We also showed cases where it is necessary to have explicit control over TCP parameters to increase the average utilization of the path or alternatively substitute a better congestion control algorithm suitable for both shared networks and light paths.

Further research is needed to look at asymmetric cases. There is still little work done in trying to maximize throughput when interoperating between NICs of different speeds over high bandwidth delay product networks; e.g., a 10 Gbps host as sender and 1 Gbit/s host as a receiver. Another area of interest is where a 10 Gbit/s host sends to a 1 Gbit/s host which already has some background traffic. In both cases, it would be interesting to explore what value of AI MD would optimize the goodput.

Acknowledgements

The transatlantic links used for conducting this research were provided to us by SURFnet, TYCO and LEVEL3. Antony Antony and Hans Blom were funded by the FP5 IST Program of the European Union

(grant IST-2001-32459, DataTAG project). Jason Lee was supported in part by the U.S. Department of Energy Contract No. DE-AC03-76SF00098. The authors would like to thank Brian Tierney and the Net100/Web100 collaborations for instrumenting TCP.

References

- [1] DataTAG Project:
<http://www.datatag.org>
- [2] DataTAG Hosts Database
<http://www.datatag.org/hosts>
- [3] J. Lee, D. Gunter, B. Tierney, W. Allcock, J. Bester, J. Bresnahan, S. Tuecke, "Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks", in Proc of Computers in High Energy Physics 2001 (CHEP 2001), Beijing, China
- [4] M. Mathis, John Heffner, "Packetization Layer Path MTU Discovery", Work in Progress, June 2004
- [5] A. Antony, J. Blom, C. de Laat, J. Lee, W. Sjouw, "Microscopic Examination of TCP Flows Over Trans-Atlantic Links", iGrid2002 special issue, FGCS, volume 19, issue 6, pp. 1017-1029

- [6] Net100 Project:
<http://www.net100.org>
- [7] iGrid2002 conference:
<http://www.igrid2002.org>
- [8] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks", ACM SIGCOMM Computer Communication Review, Vol. 33, No. 2, pp. 83-91, April 2003
- [9] R. L. Cottrell, C. Logg, I-Heng Mei "Experiences and Results from a New High Performance Network and Application Monitoring Toolkit", in Proc. of PAM 2003, April 2003.
- [10] R. L. Cottrell, A. Antony, C. Logg and J. Navratil, "iGrid2002 demonstration: Bandwidth from the low lands", technical report SLAC-PUB-9560, Stanford University, 2002.
- [11] Land Speed Record 2002,
<http://archives.internet2.edu/guest/archives/I2-NEWS/log200301/msg00005.html>
- [12] T. Hacker and B. Noble, "The Effects of Systemic Packet Loss on Aggregate TCP Flows", in Proc of IEEE Supercomputing 2002
- [13] M. Mathis, R. Reddy, J. Heffner, and J. Saperia, "TCP Extended Statistics MIB", Internet draft (Work in progress), November 2002. URL: <http://www.web100.org/mib/>
- [14] Postel J., "Transmission Control Protocol", RFC 793, IETF, September 1981,
- [15] M. Allman, V. Paxson, W.R. Stevens, "TCP Congestion Control", RFC 2581, IETF, April 1999
- [16] V. Jacobson, "Congestion Avoidance and Control", in Proc. SIGCOMM 1998, ACM Computer Communication Review, Vol. 18, No. 4, August 1988.
- [17] StarTap:
<http://www.startap.net/starlight/>
- [18] Web100 Project:
<http://www.web100.org/>
- [19] D. Katabi, M. Handley, and C. Rohrs "Congestion Control for High Bandwidth-Delay Product Networks", in Proc. of ACM Sigcomm 2002.
- [20] W. Bethel, B. Tierney, J. Lee, D. Gunter, S. Lau, "Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization", in Proc. of the IEEE Supercomputing 2000 Conference, Nov. 2000
- [21] S. Wallace, "Tsunami: A Hybrid TCP/UDP based file transfer protocol", Whitepaper available at: <http://www.ncne.nlanr.net/>
- [22] H. Sivakumar, Mazzucco, M., Zhang, Q., and R. Grossman, "Simple Available Bandwidth Utilization Library for High Speed Wide Area Networks". submitted for publication in Journal of SuperComputing, 2004
- [23] C. Jin, David Wei Steven Low "FAST TCP: Motivation, Architecture, Algorithms, and Performance", in Proc. of IEEE Infocom, March 2004
- [24] H. C. Newman, "Data Intensive Grids and Networks for High Energy and Nuclear Physics Drivers of the Formation of an Information Society", World Summit on the Information Society, Pan-European Ministerial Meeting, Bucharest, Romania, November 2002
- [25] S. Floyd, "HighSpeed TCP for Large Congestion Windows", RFC 3649, IETF, December 2003
- [26] Data Intensive Distributed Computing
<http://www-didc.lbl.gov/tuning>